Advanced Java Programming

Assignment 1

1. Write a `VendingMachine` class to simulate interactions with a vending machine. It should allow users to deposit coins with `public void deposit(int cents)`, and then to purchase a snack with `public boolean purchase(String snack)`, which should return `true` if the snack was successfully purchased and `false` if not enough money was deposited. Finally, it should support `public int getChange()` to have the machine return all money deposited. When writing this program, be sure to consider invalid user inputs and mark your states `private`.

   Note that the `VendingMachine` class needs to keep track of snacks available and their prices. This field should be `static`, so prices are constant across all `VendingMachine` objects. How you decide to keep track of this information is up to you. You can use parallel arrays of snack names and prices, or if you are familiar with Java's `HashMap` class, you can create a mapping of names to prices. You may also want to implement `public static int getPrice(String snack)` for user convenience.

2. Write a `Fraction` class to represent a fractional number. It should have two states, `numerator` and `denominator`, both of which should be marked `private` and should have accessor methods. You should have two constructors: one should accept two integers as the numerator and denominator, and the other should accept a single integer to represent a whole number. The fraction should be stored in its most reduced form. You should support the methods `public Fraction add(Fraction other)` that adds two `Fraction` objects and returns a new `Fraction` object representing the result, and should similarly support `public Fraction subtract(Fraction other)`, `public Fraction multiply(Fraction other)`, and `public Fraction divide(Fraction other)`. All of these methods should return `Fraction` objects that have been reduced as much as possible. You should also have a `public Fraction getReciprocal()` method that returns a Fraction that represents the reciprocal of this fraction. Finally, you should have a `public double toDouble()` method that converts the fraction into a floating point value.

3. Write a `MutableIntArray` wrapper class for an array of integers. The goal of this wrapper is to simulate an array of mutable length; that is, an array to which we can append and insert values and from which we can remove values.

If you are familiar with Java's `ArrayList` class, this is essentially a lightweight implementation of that (if you aren't familiar with it, we will be discussing it soon). You can support this behavior within the class by maintaining an array that is longer than the data you are currently storing. When you need to expand, allocate a new, longer array and copy the data from the old array to the new one. Likewise, you should allocate a smaller array if many items are removed and your array has a lot of empty space. Exactly when to allocate more and less space is up to you. This internal array should be marked `private`.

`MutableIntArray` should have two constructors: one should take an array of integers to supply the initial data to be copied, and another should take a size and internally create an empty array of that size. It should support the methods `public int get(int index)` to retrieve the integer stored at a given index, `public void add(int value)` to append a value to the end of the array, `public void add(int index, int value)` to insert a value at the given index, `public int remove(int index)` to remove a value at a given index and return it, `public int size()` to get the size of the current array, and `public int[] toArray()` to get an array representation of the internal data. Be sure that even if the internal array has empty space, the one returned by `toArray()` does not.